NATURE INSPIRED COMPUTING

LAB PROJECT TYPE COURSE:

IMPLEMENTING BAT ALGORITHM ON 0-1 KNAPSACK PROBLEM

MERIAM JOSEPH, 2018A7PS0291U

Table of Contents

INTRODUCTION

Nature Inspired Computing is a branch of Computer Science that strives to develop and improve computing techniques by observing how nature behaves to solve complex problems. Nature Inspired Computing has led to groundbreaking research and created new branches like neural networks, evolutionary computation, and artificial intelligence.

Swarm Optimization is a computational method that optimizes a problem by generating a population of candidate solutions and moving these candidates in a search space according to a mathematical formula. Eventually the swarm moves to the best solutions. Bat Algorithm is based on swarm intelligence heuristic search algorithm. It is a metaheuristic algorithm for global optimization. The algorithm is inspired by the echolocation of bats when hunting for prey, with varying loudness, frequency, pulse rates of emission, distance, and speed. The bat algorithm can be implemented on combinatorial optimization problems. These are problems where an optimal solution must be identified from a finite set of solutions. These solutions are either discrete or can be made discrete.

0-1 Knapsack Problem is problem of NP-Hard class. It is also a combinatorial optimization problem. Problems in NP-Hard (Non-Deterministic Polynomial-Time Hardness) classes are those which are at least as hard as the hardest problems in NP. It is not a decision problem, which can be answered with a 'Yes' or 'No'. In 0-1 knapsack problem, given a set of items, each with a different weight and price, the goal is to choose items to get the maximum possible price, while at the same not exceed the maximum weight limit of the knapsack. In 0-1 knapsack the binary numbers 0 and 1 are the decision variables that decide which item gets selected to be put into the bag.1 means the item is chosen, while 0 means the item is not chosen. Hence, the general Bat Algorithm will have to be slightly modified to discretize the random solutions that will be generated.

Nature Inspired Computing is an important field that efficiently helps implement nature inspired algorithms onto optimization problems. In this report, we will be implementing Binary Bat Algorithm to 0-1 Knapsack Problem to find the optimal solution.

LITERATURE SURVEY

In [1], Angle-modulated bat algorithm (AMBA) enables Bat Algorithm to operate in binary spaces. The paper proposes a new AMBA variant called amplitude AMBA (A- AMBA) which overcomes the limitations of the original AMBA to solve 0-1 knapsack problems. In [2], a binarization mechanism that uses the concept of percentile is applied to the bat algorithm. Additionally, the binary percentile algorithm was compared with other algorithms and produced competitive results. In [3], the entire search ability of the bat algorithm implemented on Multiple Knapsack Problem is improved by optimizing the effective solution using greedy algorithm. Hence, a Multiple Knapsack Bat Algorithm—Greedy Algorithm or MKBA-GA, for solving the MKP was proposed. Single Running Technique (SRT) was further implemented to optimize the effective solution, which gave way to the SRT-based MKBA-SRT bat algorithm. On verification, the solution ability of MKBA-GA and the MKBA-SRT algorithms were stronger than that of BBA.

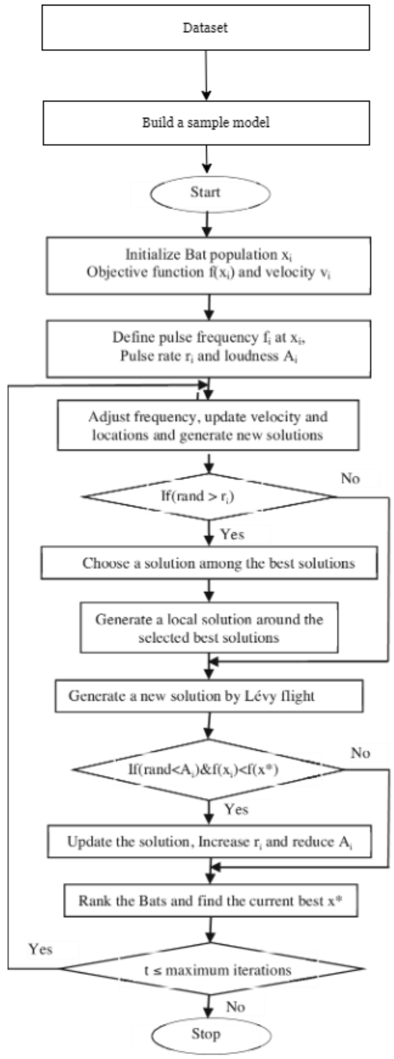
In [4], the optimal solution for 0-1 knapsack problem is obtained by implementing a binary multi-scale quantum harmonic oscillator algorithm (BMQHOA) with genetic operator. The framework of the algorithm consists of energy level stabilization, energy level decline and scale adjustment. Repair operator with greedy strategy in the algorithm guarantees the knapsack capacity constraint. Experimental results for BMQHOA give the best solutions that are accurate most knapsack data sets. In [5], the improved whale optimization algorithm (IWOA) is implemented on 0–1 knapsack problem of single dimension and multidimensions in small and large datasets. This algorithm is compared with other algorithms to validate the effectiveness in solving 0–1 knapsack problem. Furthermore, the results show that IWOA is efficient and robust for solving the hard 0–1 knapsack problem.

In [6], the paper describes problem solving approach using genetic algorithm (GA) for the 0-1 knapsack problem. The experiments start with some initial value of Knapsack variables and continue until the best value is achieved. In [7], The proposed algorithm combines local search scheme (LSS) and binary bat algorithm (BBA). The bat algorithm increases the exploration capability of the bats while LSS increases the exploitation tendencies and prevents the BBA–LSS from the entrapment in the local optima. Experimental results show that the BBA–LSS can solve 0-1 knapsack problems of larger scales.

In [8], combining bat algorithm and genetic variation, the paper introduces the processing rules of the active evolutionary operator, invalid bat and the current optimal position. This algorithm is shows better experimental results than the general bat algorithm in convergence speed and accuracy for solving the 0-1 knapsack problem. In [9], a complex-valued encoding bat algorithm (CPBA) is proposed for solving 0-1 knapsack problem. The real and imaginary parts of the complex numbers are separately updated. The algorithm effectively enhances convergence performance and diversifies bat population. CPBA improves exploration ability and is effective for solving all scales of 0-1 knapsack problem.

In [10], a comparative study is conducted on implementing algorithms like genetic algorithms, dynamic programming, branch and bound, simulated annealing, greedy search and hybrid genetic annealing for solving 0-1 knapsack problems. It resulted in branch bound dynamic programming and hybrid genetic annealing being the most efficient way to solve the problem.

METHODOLOGY



Firstly, the dataset is used to build a model where we obtain the optimal solution from the dataset. This data and solution are then used to train the fitness function.

The 0-1 Knapsack problem is mathematically written as

$$\mathrm{Max}\, f(\mathbf{x}) = \sum_{j=1}^{N} p_j x_j,$$

$$\text{s.t.}: \left\{ \begin{array}{l} \sum_{j=1}^{N} w_j x_j \le C, \\ x_j = \{0,1\}, j = 1,2,\ldots,N \\ p_j > 0,\ w_j \ge 0,\ C > 0 \end{array} \right\}$$

Where N is there number of items in the knapsack; C is the capacity of the knapsack; w_j is the weight and p_j is the price of the j^{th} item respectively; x_j are the decision variables that represent with 1 or 0 if an item is present in the knapsack or not.

We use the Binary Bat Algorithm to optimize 0-1 Knapsack problem. Binary Bat Algorithm is like that of the general Bat Algorithm, except, there is a discretization of the random solutions generated. This discretization is necessary as the prices pertaining to maximum weight of the knapsack is chosen by a decision array containing binary numbers 0 and 1. 1 means that the item is in the bag, and 0 means otherwise.

The BBA algorithm first initializes a random population of bats in an n dimensional search space. The position (x), velocity (v), frequency(α), loudness (A), pulse rate (r) all gets updated as per the Figure 1 and the following formulas: -

$$\alpha_i = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min})\beta$$

$$v_i^{t+1} = v_i^t + (x_i^t - x^{\text{best}})\alpha$$

$$x_i^{t+1} = x_i^t + v_i^{t+1},$$

$$x_{\text{new}} = x_{\text{old}} + \varepsilon A^t$$

$$r^{t+1}(i) = r^0(i) \times [1 - \mathrm{e}^{-\gamma t}]$$

$$A^{t+1}(i) = \delta A^t(i),$$

ε is a random number between [-1,1] and $\delta>0$ and $\gamma>0$ are constants.

This helps optimize the solution by picking out the best solution from every generation of bats until the optimal solution is obtained. We then compare the optimal solution obtained by the algorithm to the actual optimal to find out the accuracy of the algorithm.

EXPERIMENTAL SETUP

The experimental results are collected using the following setup. Two types of data were used for the experiment. A Large-Scale Data Consisting of 21 testcases of large-scale data, each containing thousands of data points with weight and price attributes and the integer optimal price of each of the 21 testcases. The second data used is Low Dimensional Data consisting of 10 testcases of large-scale data, each containing thousands of data points with weight and price attributes. It also contains the integer optimal price of each of the 10 testcases. The program was coded using Python language and libraries like NumPy, pandas, OS and matplotlib using the software Visual Studio Code. We then clean the data in a way that helps the program make use of it efficiently.

EXECUTABLE CODE

```
#importing the necessary modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from os import listdir
from os.path import isfile, join

#Function to import data from file in proper format
def import_data():

    #import data files
    ls_data= 'large_scale/'
    ls_optimum= 'large_scale-optimum/'
    ld_data= 'low-dimensional/'
    ld_optimum= 'low-dimensional-optimum/'

    #isfile returns true if file exists
    #listdir lists all the directories in the path
    #join joins one or more path intelligently
    ls_files = [f for f in listdir(ls_data) if isfile(join(ls_optimum, f))]
    ld_files = [f for f in listdir(ld_data) if isfile(join(ld_optimum, f))]

    #create arrays to store D=datapoints, O=optimum values, M=maximum weights
    ls_D = []
    ls_O = []
    ls_M = []
    for filename in ls_files :
        #read data excluding last line (which is the answer)
        x = pd.read_csv(ls_data + filename,names=['weight', 'price'], delimiter = ' ', header=None,error_bad_lines=False )
        max_w = x.iloc[0]['price']
        ls_M.append(max_w)
        ls_D.append(x.iloc[1:-1])
        y =  pd.read_csv(ls_optimum+filename,names=['optimal'], delimiter = ' ', header=None)
        ls_O.append(y.iloc[0]['optimal'])

```

```
    ld_D = []
    ld_O = []
    ld_M = []
    for filename in ld_files :
        #read data excluding last line (wich is the answer)
        x = pd.read_csv(ld_data+filename,names=['weight', 'price'], delimiter = ' ', header=None,error_bad_lines=False )
        max_w = x.iloc[0]['price']
        ld_M.append(max_w)
        ld_D.append(x.iloc[1:])
        y =  pd.read_csv(ld_optimum+filename,names=['optimal'], delimiter = ' ', header=None)
        ld_O.append(y.iloc[0]['optimal'])

    return ls_D, ls_O, ls_M, ld_D, ld_O, ld_M

#LS = data large scale, consists of hundreds of datapoints
#LD = data low dimension, has less than 100 datapoints
LS_X, LS_Y, LS_W, LD_X, LD_Y, LD_W = import_data()

#select LD first to build up the model (faster execution)
data_w = LD_W[0] #maximum weight
data_x = LD_X[0] #datapoints
data_y = LD_Y[0] #optimum values
print("Data 1:")
print("Items given are:")
print(data_x)
print("Maximum Weight that Knapsack can contain:")
print(data_w)
print("Optimal Value is:")
print(data_y)


#generate a random solution by choosing items using values between 0 and 1
sol = np.random.random(len(data_x))
print("\nArray that represents a random solution:")
print(sol)
```

```python

#discretization so solution array contains only 0 or 1 values
print("Discretization of random solution:")
print(np.round(sol).astype(bool))


#display info of selected items in a table
idx = np.arange(len(data_x))
idx = idx[np.round(sol).astype(bool)]
carried = data_x.iloc[idx]
print("\nTable of selected items based on the random solution generated")
print(carried)

#Fitness Function
def fit(sol, data, max_w) :
    #Discretization
    mask = np.round(sol).astype(bool)

    #decode
    idx = np.arange(len(data))
    idx = idx[mask]
    data_solution = data.iloc[idx] #arranges solution in a table to view


    #price
    price = np.sum(data_solution['price'])
    weight = np.sum(data_solution['weight'])
    if weight <= max_w:
        return price
    else :
        return 0


```

```python
class bat:
    def __init__(self, population, data, max_w, fmin, fmax, A, alpha, gamma):
        self.population = population #population
        self.data = data #data
        self.max_w = max_w #maximum weight
        self.fmin = fmin #minimim frequecy
        self.fmax = fmax #maximum frequecy
        self.A = A #loudness
        self.alpha = alpha #pulse frequency
        self.gamma = gamma #coefficent of pulse emission
        self.data_size = len(data)
        self.best_sol = None
        self.t = 1 #iteration

        #to generate random values to bats
        self.init_x()
        self.init_f()
        self.init_v()
        self.init_y()
        self.init_r()

    def init_x (self): #to generate random positions for bats
        self.solutions =  np.random.random((self.population,self.data_size)) # (self.population * self.data_size) array of random numbers

    def init_f (self):#to generate random frequencies for bats
        self.f = np.random.uniform(self.fmin,self.fmax,self.population) #give uniform self.population no.of random numbers between fmin and fmax

    def init_v (self): #to generate random velocities for bats
        self.v = np.zeros((self.population, self.data_size)) #zero array of (self.population * self.data_size)

    def init_y (self):
        Y = np.zeros(len(self.solutions)) #zero array of self.solutions length
        for i,sol in enumerate(self.solutions) :
            Y[i] = fit(sol,self.data, self.max_w)  #to find current solution

        self.Y = Y
```

```python
    def init_r (self): #generate random pulse rates
        self.r = np.random.random(self.population)
        self.r0 = self.r


    def update_f(self): #to update frequency of bats
        self.fmin = np.min(self.f)
        self.fmax = np.max(self.f)
        betha = np.random.random(len(self.f))
        self.f = betha*(self.fmax-self.fmin) + self.fmin

    def update_v(self): #to update velocity of bats
        self.find_best_solution()
        r = (self.solutions - self.best_sol)
        rr = [r[i] * self.f[i] for i in range(len(r))]

        self.v = self.v + rr
        self.normalize_v()

    def update_x(self): #update position of bats
        self.solutions += self.v
        self.normalize_solution()
        self.update_y()
        self.localsearch()
        self.update_y
        self.find_best_solution()

    def update_A(self): #update loudness
        self.A = self.A * self.alpha

    def update_r(self): #update pulse rate
        self.r = self.r0 * (1- np.exp(-self.gamma*self.t))
        self.t += 1

```

```python
    def update_y(self): #to update the current solution
        Y = np.zeros(len(self.solutions))
        for i,sol in enumerate(self.solutions) :
            Y[i] = fit(sol,self.data, self.max_w)
        self.Y = Y


    def find_best_solution(self): #to find best solution
        self.best_sol = self.solutions[np.argmax(self.Y)]

    def normalize_solution(self):  #to normalize the solution
#          self.solutions = np.absolute(self.solutions / (np.max(self.solutions) - np.min(self.solutions)))
        self.solutions[self.solutions > 1] = 1
        self.solutions[self.solutions < 0] = 0

    def normalize_v(self):
        self.v = np.sin(self.v) #sin value of v

    def extract_solution(self):
        #Discretization
        mask = np.round(self.best_sol).astype(bool) #round off the solution

        #decode
        idx = np.arange(len(self.data))
        idx = idx[mask]
        data_solution = self.data.iloc[idx]

        return data_solution

    def mutate(self,x):
        size = len(x)
        sizex = size//5
        idx = np.random.permutation(size)[:sizex]

        x[idx] = 1-x[idx]
```

```
        return x

    def localsearch(self):
        idxm = np.where(self.Y == 0)
        cm = self.solutions[ idxm ]
        for i in range(len(cm)):
            cm[i] = self.mutate(cm[i])

        self.solutions[idxm] = cm



# Bat Algorithm parameters
fmin = 0
fmax = 1
A = 1
alpha = 0.98
gamma = 0.98
population = 75
epoch = 25 #generation


#Selecting DLD
data_w = LS_W[4] #maximum weight
data_x = LS_X[4] #datapoints
data_y = LS_Y[4] #optimum values
print("Data 2:")
print("Items given are:")
print(data_x)
print("Maximum Weight that Knapsack can contain:")
print(data_w)
print("Optimal Value is:")
print(data_y)

```

```
data_w = LD_W[8] #maximum weight
data_x = LD_X[8] #datapoints
data_y = LD_Y[8] #optimum values
print("Data 3:")
print("Items given are:")
print(data_x)
print("Maximum Weight that Knapsack can contain:")
print(data_w)
print("Optimal Value is:")
print(data_y)


solution = []
acc = []
bat_behaviour = []
for loop in range(30):
    localSolution = []
    bats = bat(population,data_x,data_w,fmin,fmax,A,alpha,gamma)
    for i in range(epoch):
        bats.update_f()
        bats.update_v()
        bats.update_x()
        bats.update_r()
        localSolution.append(np.max(bats.Y))
#          if (i==epoch-1):
    bat_behaviour.append(np.average(bats.Y))
    solution.append(sum(localSolution)/len(localSolution))
    acc.append((sum(localSolution)/len(localSolution))/data_y)

averageSolution = sum(solution)/len(solution)
```

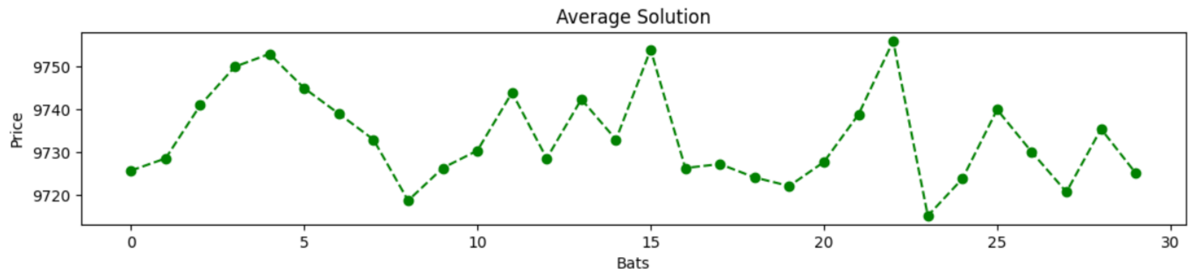
```
plt.figure(figsize=(13,5))
plt.figure(1)
plt.subplot(211)
plt.plot(solution, "go--")
plt.xlabel("Bats")
plt.ylabel("Price")
plt.title('Average Solution')
averageAcc = (sum(acc)/len(solution))*100
print ("Average accuracy of Data 3 equal to",'%.3f' % averageAcc, ' %')
print ("with an average solution equal to",'%.3f' % averageSolution, " from ", data_y)
plt.show()

plt.figure(figsize=(13,5))
plt.subplot(212)
plt.plot(bat_behaviour, "ro--")
plt.title('Average Behaviour of Bat')
plt.xlabel("Bats")
plt.ylabel("Price")
plt.show()
print ("with an average behaviour of bat colony equal to",'%.3f' % np.average(bat_behaviour))
```

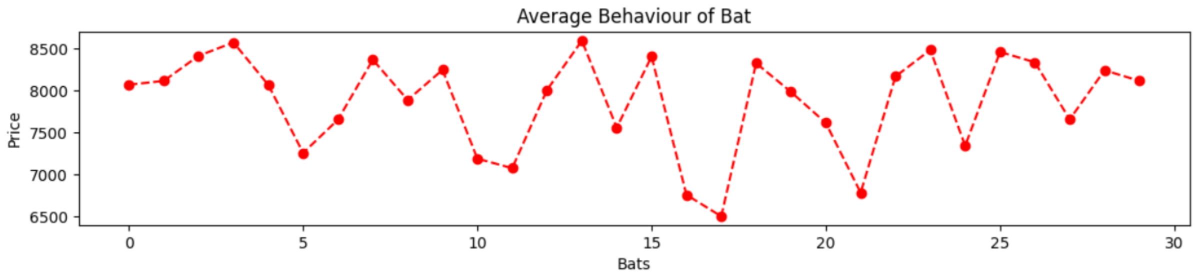
EXPERIMENTAL RESULTS

```
Data 1:
Items given are:
    weight  price
1       91     84
2       72     83
3       90     43
4       46      4
5       55     44
6        8      6
7       35     82
8       75     92
9       61     25
10      15     83
11      77     56
12      40     18
13      63     58
14      75     14
15      29     48
16      75     70
17      17     96
18      78     32
19      40     68
20      44     92
Maximum Weight that Knapsack can contain:
879
Optimal Value is:
1025

Array that represents a random solution:
[0.80896242 0.21562839 0.62151328 0.16022775 0.31793174 0.0722095
 0.26794033 0.85937004 0.72755506 0.87969267 0.19467854 0.95830534
 0.98130221 0.86781147 0.25620328 0.31440482 0.42301406 0.11348917
 0.14010513 0.66878213]
Discretization of random solution:
[ True False  True False False False False  True  True  True False  True
  True  True False False False False False  True]
```

```
Table of selected items based on the random solution generated
    weight  price
1       91     84
3       90     43
8       75     92
9       61     25
10      15     83
12      40     18
13      63     58
14      75     14
20      44     92
Data 2:
Items given are:
     weight  price
1        94    485
2       506    326
3       416    248
4       992    421
5       649    322
..      ...    ...
196      25    635
197     380    225
198     712    823
199     266    164
200     216    343

[200 rows x 2 columns]
Maximum Weight that Knapsack can contain:
1008
Optimal Value is:
11238
```

```
Data 3:
Items given are:
    weight  price
1      981    983
2      980    982
3      979    981
4      978    980
5      977    979
6      976    978
7      487    488
8      974    976
9      970    972
10     485    486
11     485    486
12     970    972
13     970    972
14     484    485
15     484    485
16     976    969
17     974    966
18     482    483
19     962    964
20     961    963
21     959    961
22     958    958
23     857    959
Maximum Weight that Knapsack can contain:
10000
Optimal Value is:
9767
Average accuracy of Data 3 equal to 99.710  %
with an average solution equal to 9738.675  from  9767
with an average behaviour of bat colony equal to 7988.677
```





CONCLUSION

Nature provides solutions to many of the problems that we have today. We can take inspiration from nature for developing problem solving techniques. The Bat Algorithm is one of many nature inspired algorithms (Ant Colony, Grey Wolf, Genetic) that help solve optimization problems, specifically combinatorial optimization problems.

In this report, we have successfully implemented Bat Algorithm to find the optimal solution to 0/1 knapsack problem with a 99.71% accuracy.

REFERENCE

1. Huang, X., Li, P., & Pu, Y. (2019). Amplitude Angle Modulated Bat Algorithm with Application to Zero-One Knapsack Problem. IEEE Access, 7, 27957–27969.

2. Jorquera, L., Villavicencio, G., Causa, L., Lopez, L., & Fernández, A. (2020). A Binary Bat Algorithm Applied to Knapsack Problem. Advances in Intelligent Systems and Computing, 172–182.

3. Li, S., Cai, S., Sun, R., Yuan, G., Chen, Z., & Shi, X. (2019). Improved Bat Algorithm for Multiple Knapsack Problems. Computer Supported Cooperative Work and Social Computing, 143–157.

4. Huang, Y., Wang, P., Li, J., Chen, X., & Li, T. (2019). A Binary Multi-Scale Quantum Harmonic Oscillator Algorithm for 0–1 Knapsack Problem with Genetic Operator. IEEE Access, 7, 137251–137265.

5. Abdel-Basset, M., El-Shahat, D., & Sangaiah, A. K. (2017). A modified nature inspired meta-heuristic whale optimization algorithm for solving 0–1 knapsack problem. International Journal of Machine Learning and Cybernetics, 10(3), 495–514

6. Saraswat, Manish and Tripathi, Ramesh Chandra, Solving Knapsack Problem with Genetic Algorithm Approach (April 1, 2020). Knapsack Problem with Genetic algorithm Approach. Proceedings of the International Conference on Innovative Computing & Communications (ICICC) 2020.

7. Rizk-Allah, R. M., & Hassanien, A. E. (2017). New binary bat algorithm for solving 0–1 knapsack problem. Complex & Intelligent Systems, 4(1), 31–53.

8. Chen, Y. (2016). A Novel Bat algorithm of solving 0-1 Knapsack Problem. Proceedings of the 2016 4th International Conference on Machinery, Materials and Computing Technology

9. Zhou, Y., Li, L., & Ma, M. (2015). A Complex-valued Encoding Bat Algorithm for Solving 0–1 Knapsack Problem. Neural Processing Letters, 44(2), 407–430.

10. A. E. Ezugwu, V. Pillay, D. Hirasen, K. Sivanarain and M. Govender, (2019) "A Comparative Study of Meta-Heuristic Optimization Algorithms for 0 – 1 Knapsack Problem: Some Initial Results," in IEEE Access, vol. 7, pp. 43979-44001